Simulation Modes: Statevector, Density Matrix, Aer

Variational Quantum Algorithms · Module 9 — Hardware Noise Models

Node 9.7

Position in Knowledge Graph

System: Variational Quantum Algorithms **Structural Domain:** Hardware Noise Models **Node:** 9.7

Before you run a VQA on real hardware, you almost always simulate it classically. Simulation serves several purposes: developing and debugging the algorithm, evaluating expressivity and trainability, testing noise mitigation, and producing baselines against which to compare the hardware results. Getting simulation right is foundational to the whole VQA research enterprise.

But there isn't one kind of quantum simulation — there are several, each with different capabilities, costs, and use cases. The main modes:

1. **Statevector simulation** — exact, noiseless, stores the full 2^n-dimensional amplitude vector. Fastest for small n, exponentially expensive.
2. **Density matrix simulation** — exact, supports mixed states and general noise channels, stores a $2^n \times 2^n$ matrix. Doubles the memory cost compared to statevector.
3. **Stochastic / trajectory simulation (Qiskit Aer, quantum trajectory methods)** — Monte Carlo sampling of pure-state trajectories under noise. Memory like statevector but with a shot overhead for noise averaging.
4. **Tensor network simulation (MPS, PEPS)** — exact or approximate, uses low-entanglement structure. Scales better for certain classes of circuits.
5. **Clifford simulation (stabilizer formalism)** — exact and efficient for Clifford-only circuits. Limited to Clifford gates + Pauli observables.
6. **Approximate classical simulation** — methods that produce an approximation of the state with a budget.

This node covers the main modes relevant to VQAs: statevector (the fastest for small n), density matrix (the most expressive noise model), Aer-style stochastic trajectories (the practical middle ground), and tensor networks (the scaling solution for structured states). Understanding the tradeoffs is essential for designing VQA experiments that can actually be simulated within a reasonable compute budget.

Statevector Simulation

The simplest mode: store the amplitude vector $|\psi\rangle \in \mathbb{C}^{2^n}$ directly, and apply each gate as a matrix-vector multiplication.

Memory. $2^n \times 16$ bytes (complex doubles) $= 2^{n+4}$ bytes. For $n = 30$, that's 16 GB (feasible on a workstation). For $n = 35$, 512 GB (big machine). For $n = 40$, 16 TB (supercomputer). The effective limit is around $n = 30 - 35$ on commodity hardware.

Compute. Each gate is an $O(2^n)$ operation (for single-qubit gates) or $O(2^n)$ for two-qubit gates (with careful implementation). A depth-L circuit costs $O(L \cdot 2^n)$.

Noise handling. Statevector simulation is *noiseless*. If you want to simulate noise, you need to either switch to a different mode (density matrix, trajectory) or inject the noise as classical randomness (e.g., stochastically

applying Pauli errors between gates).

Pros: fastest for small n, simplest code, no approximation.

Cons: exponential in n, no native noise support.

Use cases for VQAs: - Developing and debugging the algorithm structure. - Computing exact expectation values to serve as ground truth for noisier simulations. - Evaluating expressibility (via frame potential sampling) and trainability (via exact gradient variance). - Small-scale proof-of-concept experiments.

Qiskit's `AerSimulator(method='statevector')`, Cirq's `Simulator`, PennyLane's `default.qubit`, and Intel QS are typical implementations.

Density Matrix Simulation

Stores the full density matrix $\rho \in \mathbb{C}^{2^n \times 2^n}$. This is the most general state representation — it can handle mixed states, which are required for anything involving noise, partial traces, or decoherence.

Memory. $2^{2n} \times 16$ bytes. For $n = 15$, 16 GB. For $n = 18$, 1 TB. Effective limit is $n \approx 15$ on a workstation.

Compute. Each gate application is a superoperator action on ρ, costing $O(4^n)$ naively but can be made $O(2^n \cdot k)$ for k-local gates with smart implementation. Noise channels are applied the same way as gates, using Kraus operators.

Pros: full generality, exact noise simulation with any Kraus channel, clean handling of partial traces and reduced states.

Cons: memory cost is squared compared to statevector — half the qubit limit ($n \approx 15$ vs 30).

Use cases for VQAs: - Testing noise mitigation techniques on small systems. - Comparing different noise models (depolarizing vs amplitude damping). - Computing exact noisy gradients and their variances. - Validating noise-aware ansatz designs.

When to use. When you need to simulate noise and your system is small enough to afford the memory cost.

Qiskit's `AerSimulator(method='density_matrix')` and `qiskit_aer.noise.NoiseModel` is the standard implementation.

Stochastic Trajectory Simulation

The practical middle ground: use pure-state (statevector) simulation, but randomly inject noise at each gate according to the noise channel's Kraus decomposition.

How it works. For each gate followed by a noise channel with Kraus operators $\{K_k\}$, sample a k with probability $\|K_k|\psi\rangle\|^2$ and apply $K_k|\psi\rangle / \|K_k|\psi\rangle\|$. Repeat many times and average.

Memory. Same as statevector: $O(2^n)$. Can simulate up to $n \approx 30$.

Compute. Same as statevector per trajectory, but you need many trajectories to average out the noise. Typically 100-10000 trajectories per data point.

Pros: scales to large n, can handle any Kraus channel, no exponential blowup from density matrix.

Cons: each trajectory is noisy — the simulated observable estimate has variance from *both* the original shot noise *and* the Monte Carlo noise averaging. Can require many more trajectories than naive shot count.

Use cases for VQAs: - Simulating noisy VQAs on 20-30 qubits with realistic noise. - Testing mitigation techniques at scale. - Producing the baseline "noisy simulation" against which hardware runs are compared.

When to use. When you need noisy simulation *and* density matrix is too expensive (typically $n > 15$).

Qiskit Aer's `AerSimulator(method='matrix_product_state')` and `method='stabilizer'` are not trajectory methods, but the default `method='automatic'` often uses trajectory methods when noise is present. PennyLane's `default.mixed` and Cirq's `density_matrix_simulator` are alternatives.

Tensor Network Simulation (MPS, PEPS)

For quantum states with *low entanglement*, tensor network methods scale much better than the general-state methods.

Matrix Product States (MPS). Write the state as

$$|\psi\rangle = \sum_{i_1,\ldots,i_n} \mathrm{Tr}(A_1^{i_1} A_2^{i_2} \cdots A_n^{i_n}) |i_1 \cdots i_n\rangle,$$

where $A_k^{i_k}$ are matrices of dimension $\chi \times \chi$ (called the "bond dimension").

Memory. $O(n \cdot \chi^2)$. Grows polynomially in n and in χ.

Compute. Each two-qubit gate costs $O(n \cdot \chi^3)$ in the worst case.

Key observation. The bond dimension χ is related to the *entanglement* in the state: it must be at least $2^{S/\ln 2}$ where S is the entanglement entropy between a subsystem. For *low-entanglement states*, χ stays small and simulation is efficient.

Pros: can simulate very large systems (hundreds of qubits) for low-entanglement states. Handles noise via purification (adding ancilla legs).

Cons: fails for highly entangled states — the bond dimension explodes. Also awkward for non-local interactions (two-qubit gates between distant qubits).

Use cases for VQAs: - Benchmarking VQAs on large systems (50-200 qubits) when the state remains low-entanglement. - Testing circuits on lattice Hamiltonians where the ansatz preserves locality. - Dequantization studies — seeing if the quantum state could be simulated classically.

When *not* to use. For highly entangled circuits (random HEA, deep QAOA), MPS fails. Statevector or trajectory is the better choice.

Reference. Vidal (2004) introduced time-evolving block decimation (TEBD); modern packages include ITensor, quimb, and TeNPy.

2D generalization (PEPS). Projected Entangled Pair States work on 2D lattices. Useful for 2D VQAs but have their own convergence issues.

Clifford Simulation

A very different mode: the **stabilizer formalism** tracks quantum states as collections of "stabilizers" (Pauli operators that the state is an eigenstate of) rather than amplitudes.

Gottesman-Knill theorem. Any quantum circuit consisting of: - Clifford gates (CNOT, H, S, their conjugates) - Pauli measurements - Preparation in computational basis states

can be simulated classically in polynomial time — specifically, $O(n^3)$ per gate for n qubits.

Pros: exponentially faster than statevector for Clifford-only circuits. Can simulate thousands of qubits.

Cons: *only* for Clifford circuits. Any non-Clifford gate (T, parameterized rotations) breaks the formalism. Can be extended to "T-doped" circuits with $O(2^t)$ cost for t T gates.

Use cases for VQAs: - Testing the Clifford variants of a VQA (e.g., a VQA reduced to Clifford for benchmarking). - Clifford Data Regression (node 9.4) uses this as a subroutine. - Noise characterization via randomized benchmarking (Clifford circuits with varied lengths).

When to use. Whenever you can reduce your VQA to a Clifford subcircuit, use Clifford simulation for that part.

Qiskit Aer's `method='stabilizer'`, Cirq's Clifford simulator, and Aaronson's CHP are standard implementations.

Approximate Classical Simulation

Several methods produce *approximate* classical simulations of quantum states at controlled accuracy:

1. Truncated amplitude methods. Only track the largest amplitudes, discard small ones. Works when the state is sparse in the computational basis.

2. Classical shadows (Huang-Kueng-Preskill 2020). Measure the state in random bases and use the measurement record to compute arbitrary k-local observables. Not a full simulation but sufficient for VQA expectation values.

3. Slicing methods. Break the circuit into smaller slices that are simulated independently and then combined. Scales with circuit depth but with cost reduction from parallelism.

4. Machine-learning surrogates. Train a neural network to approximate the expectation value function $\boldsymbol{\theta} \mapsto C(\boldsymbol{\theta})$ from a small number of exact evaluations. Useful for exploring the landscape without running the simulator.

Use cases. These are for extreme scaling or exploration. Not primary simulation modes but complementary tools.

Simulation Mode Decision Tree

For a VQA practitioner, the choice of simulation mode depends on:

1. **System size n.**
   - $n \leq 15$: density matrix is affordable, use it for noise simulation. Statevector for noiseless.
   - $15 < n \leq 30$: statevector for noiseless, stochastic trajectory for noisy.
   - $n > 30$: tensor network if low entanglement, classical shadows for expectation values, or restrict to Clifford variants.
2. **Noise model needed?**
   - No noise: statevector, tensor network (if large).
   - Depolarizing only: stochastic trajectory is cheap.
   - General noise model: density matrix (if small enough), trajectory (if large).
3. **What's being computed?**
   - Full state: density matrix or statevector.
   - Expectation values only: classical shadows can work.
   - Gradient: parameter shift rule with any underlying simulator.
4. **Expressibility/trainability study?**
   - Noiseless statevector — no noise needed for the structural analysis.
5. **Mitigation validation?**
   - Density matrix for small, trajectory for medium, unavailable for very large.

Rule of thumb. Start with the cheapest mode that answers your question, and only upgrade when the cheaper mode is insufficient. Noise simulation is expensive; avoid it when possible.

Thesis View: Simulation Modes And The Objective Generator

From the thesis framework:

1. **The simulation mode *is* the objective generator for classical pretraining.** Before deploying to hardware, the VQA's cost function is computed via simulation; the simulation mode determines the accuracy and speed of that computation.
2. **Different simulation modes give different signal-to-noise tradeoffs.** Exact (statevector) simulation gives perfect signal; trajectory simulation matches hardware noise. Choosing the mode is a design decision that affects the training dynamics.
3. **HOPSO can use different simulators for different purposes.** Exact for expressibility checks, trajectory for realistic cost evaluation, classical shadow for large-scale exploration. The separation of objective and dynamics lets you swap simulators without changing the optimizer.
4. **Simulation is the "oracle" for the optimizer.** The optimizer doesn't know whether its cost evaluations come from hardware or simulation — as long as the interface is the same, the optimizer works. This is the practical manifestation of the thesis's abstraction.

Structural Role

This node is the **practical simulation reference** for VQA research. It covers the main simulation modes, their costs, and when to use each.

It is the bridge between the theoretical noise models (9.1-9.3) and the practical experiments that use them. Module 9's material is only useful if you can actually simulate it, and this node is the guide to doing so.

Relations to Other Concepts

- **Gottesman, Knill (1998)** — stabilizer formalism and Clifford simulation.
- **Vidal (2004)** — TEBD and MPS for quantum simulation.
- **Aaronson, Gottesman (2004)** — improved Clifford simulation.
- **Huang, Kueng, Preskill (2020)** — classical shadows.
- **Qiskit Aer documentation** — the reference implementation.
- **Module 9.1 (Quantum noise channels)** — what the simulator needs to represent.
- **Module 9.4 (Mitigation)** — what the simulator is used to test.

Worked Example — Choosing A Simulator For A 20-Qubit VQE

Problem. A 20-qubit chemistry VQE on LiH with a hardware-efficient ansatz at depth 10. Goal: evaluate the exact noiseless cost and a noisy cost with 99.5% two-qubit gate fidelity, 99.9% single-qubit fidelity, 97% readout fidelity.

Option 1 — Statevector, noiseless. - Memory: $2^{20} \cdot 16$ bytes = 16 MB. Trivial. - Compute: $10 \cdot 20 \cdot 2^{20} \approx 2 \times 10^8$ operations per data point. Fast (seconds). - Use: for the noiseless baseline.

Option 2 — Density matrix, noisy. - Memory: $2^{40} \cdot 16$ bytes = 16 TB. Unfeasible on workstation. - Skip this option.

Option 3 — Stochastic trajectory, noisy. - Memory: $2^{20} \cdot 16$ bytes = 16 MB per trajectory. - Compute: same as statevector per trajectory. Need ~1000 trajectories per data point. - Total compute: 1000 × (statevector cost) $\approx 2 \times 10^{11}$ operations ≈ minutes per data point. - Use: for the noisy baseline.

Option 4 — Hardware. - Compute: 1-10 minutes per cost evaluation including queue time. - Noise level: realistic. - Cost: real compute time and shot budget.

Workflow. Use Option 1 for landscape exploration and hyperparameter tuning; Option 3 for validating noise mitigation; Option 4 for the final runs.

Observation. For 20-qubit VQEs, density matrix simulation is already out of reach. Trajectory methods become essential. This is typical for realistic VQA experiments.

Visualization

Pending. A plot comparing simulation cost (memory, time) vs qubit count for each mode: statevector, density matrix, trajectory, MPS (at fixed bond dimension), Clifford. Log-log scale. Caption: "Simulation modes scale differently. Pick the right one."

Exercises

1. Compute the memory cost of density matrix simulation for $n = 10, 15, 20, 25$ qubits. At what point does it exceed your workstation's RAM?
2. For stochastic trajectory simulation, show that the variance of the noisy expectation value estimate is proportional to $1/N_{\text{traj}}$ and derive the constant.
3. (VQA) For a 25-qubit MPS simulation of a depth-20 HEA, estimate the bond dimension needed for 1% accuracy. Is this tractable?

Research Extensions

- **Hybrid simulation modes** — switching between simulators during a VQA run based on resource availability.
- **GPU-accelerated statevector simulation** — scaling statevector beyond $n = 30$ using multi-GPU clusters.
- **Smart noise injection** — reducing the trajectory count needed by stratified sampling.
- **Approximate density matrix methods** — low-rank or tensorized density matrices for $n > 15$.
- **Machine-learning surrogates for cost landscapes** — neural networks replacing expensive simulation during optimization.

Cross-links to Other Courses

- **Classical Computing and Algorithms** — cost models, memory hierarchy, parallelization.
- **Tensor Network Methods in Physics** — MPS, PEPS, DMRG.
- **Module 9.1 (Quantum noise channels)** — what simulators represent.
- **Module 9.4 (Mitigation techniques)** — what simulators are used to validate.
- **Module 8.7 (Quantum advantage criteria)** — the line between tractable and intractable simulation.

Variational Quantum Algorithms · Module 9 — Hardware Noise Models · Node 9.7
Concepts: density-matrix, simulation, hilbert-space, entanglement

Prerequisites: 9.1, 9.2
Enables: 9.8

hbar.university — own.your.cognition